Software Testing Report

"Mathochist Studios" Cohort 4, Team 11

Euan Cottam
Charlie Thoo-Tinsley
Harri Thorman
Will King
Zach Moussallati
Aiden Turner
Marcus Williamson
Joshua Zacek

Testing Methods and Approaches

In order to minimise errors in the game, we ran extensive tests on the game throughout development. We chose two different methods of testing, where we automatically tested as much as we could using unit tests, then manually tested other elements of the game, like rendering and tests against our requirements using user tests and QA tests within the group.
Unit Tests - Writing automated unit tests was highly appropriate in our case, since these automated unit tests can be run in the CI pipeline, we can detect errors in the logic early, and multiple developers could then work on the project (refactoring and upgrading, etc.), while being sure that the code still runs as expected.
For our unit tests, we wrote the tests using:

- JUnit: A Java testing framework that made it easier to write unit tests
- Mockito: A Java Mocking framework that makes it easier to mock objects in the game for testing purposes. During testing, this was primarily used to mock the rendering backend and insert keystrokes into the game.
- Jacoco: A code coverage library that analyses your tests and code to give you a report on how much of the functionality has been tested. This was vital when writing our tests, so we could ensure that all code that could be tested was covered.

Our automated testing reports can be found on the website
Manual Tests - These were also essential in ensuring the game remained playable and accurate to our requirements, things that can't be automated. Using manual tests, we successfully removed many soft locks in the game, and we could end-to-end test the UI/UX logic where automated tests were not applicable.

Testing Report

Parts of the system where manual tests were carried out / unit tests were not applicable:

- **Softlocks/Design errors**: A softlock is when the player gets stuck or cannot progress/finish the game due to actions of the player
    - One softlock that we fixed using manual tests was getting boxed in by the friend entity. At this point, the friend entity had a collision box, which meant that if you positioned the player in a corner, you could get boxed in and be unable to move. We fixed this by removing the friend's collision box.
    - Another softlock we found was when you collected coins in the library, then spent them all at the vending machine, you wouldn't be able to complete 100% of the game. We felt this wasn't in the spirit of the game and marked it as a softlock. We fixed this by positioning coins around the game so you could always finish the game with 100% if you chose to.
- **UI/UX Design and Human Interaction**: A UI/UX error would be when a large number of users do not understand what to do, or get stuck in the game due to the interface/controls/game design.
    - One example of this was the wallet hidden event in the library. We found a lot of people didn't know what to do with the wallet and ended up walking around with it all game. We fixed these forms of issues by adding a notification system to the game, which hinted to the player what was happening (e.g.

"You need to hand the wallet into the receptionist.", or "Going out of range of the friend makes him stop following you as you are supposed to be helping him along.")
    - Another example of a UX error we had was that a lot of people didn't know what to do at the start of the game, or forgot what they were supposed to be doing. We helped players along by adding a quest system that logged the next objective when they encountered a door or an event. We felt this was a good compromise where we don't take away from the excitement of exploration through the game, but the players also have some help once they encounter an event.
- **Rendering**: An error in rendering is difficult to test for and not appropriate for unit tests; hence was tested for through manual testing.

How every requirement was tested against:

Requirements Testing		
Requirement ID	**Test**	**Test Outcome and Measurement**
UR_POSITIVE_EVENTS, FR_POSITIVE_VENTS	We have unit tests for every event, including all positive event tests, and we have unit tests for validating the event counter as seen in EventsTests.java	All unit tests passed, all positive events counted: speed powerup, rollerskates powerup, energy drink powerup (3 positive events)
UR_NEGATIVE_EVENTS, FR_NEGATIVE_VENTS	We have unit tests for every event, including all negative event tests, and we have unit tests for validating the event counter as seen in EventsTests.java	All unit tests passed, all negative events counted: library card collection, basement key collection, slowed by water/mud, friend helping, moving ducks (5 negative events)
UR_HIDDEN_EVENTS, FR_HIDDEN_EVENTS	We have unit tests for every event, including all hidden event tests, and we have unit tests for validating the event counter as seen in EventsTests.java	All unit tests passed, all hidden events counted: hand in wallet, caught by dean, long boi pet (3 hidden events)
UR_SOUND_AND_MUSIC, FR_SOUND_AND_MUSIC		
UR_ADJUSTABLE_VOLUME, FR_ADJUSTABLE_VOLUME		
UR_FAMILY_FRIENDLY, NFR_NO_GRAPHIC_CONTENT	We asked for comments from user testing. We had parents of young children play the game and asked if they thought it was suitable for their children to play the game.	We did not use blood, gore, or 18+/adult depictions in the game. We kept colours light and fun, and took inspiration

Requirements Testing		
	We also checked to make sure that the game adhered to the BBFC PG rating guidelines.	from family-friendly games like Stardew Valley in our game design. We also tried to design the game for a BBFC PG rating (BBFC)
UR_COLOURBLIND_ACCESSIBLE, NFR_COLOURBLIND_FRIENDLY	We placed a monochrome shader over the game to block out all colour and asked for comments from user testing.	We did not use any colour-dependent signalling in the game, and colours that were used were kept purely for aesthetics. More than 80% of users reported that it was accessible
UR_DYSLEXIA_ACCESSIBLE, UR_ALWAYS_LEGIBLE_TEXT	We performed user testing with people with dyslexia and asked them to comment on how easy it was to understand the game.	We chose a font that was easy to see and understand, and kept the font large and on screen for a minimum of 5 seconds. More than 95% of users reported that it was accessible
UR_TIME_LIMIT, FR_TIME_LIMIT	We included a unit test to confirm all gameplay ended at 5 minutes. See PlayerTests.java	All unit tests passed.
UR_MAZE_LIKE, FR_MAZE_LIKE	We asked users in user testing how much the game felt like a maze	We received positive results where people said it felt like a maze, but was not too overwhelming, with nice additions like long boi
UR_SCORE_SYSTEM, FR_SCORE_SYSTEM	We wrote a unit test to test different changes to the game state and how it affected the final score calculation. See PlayerTests.java	All unit tests passed.
UR_DEAN_CHASING, FR_DEAN_CHASING	The dean was implemented to chase you in the basement after you had collected the library key. This was tested with unit tests to trigger the dean, see EventsTests.java. For tests on the dean-following-algorithm, see EnemyAITests.java	All unit tests passed.
UR_ACHIEVEMENTS, FR_ACHIEVEMENTS	We wrote a unit test to test different changes to the game state/achievements and how they affected the final score calculation. See PlayerTests.java. We also ran manual tests to check achievements run in normal gameplay	All unit tests passed. We added achievements to the notification system so people would know when they get an achievement
UR_LEADERBOARD, FR_LEADERBOARD		

Requirements Testing		
UR_VARIABLE_DIFFICULTIES, FR_VARIABLE_DIFFICULTY	Unit tests were written to change the difficulty of the game and check that the score changes accordingly. We also ran manual tests to check that the difficulty could be changed in settings and was reflected in the difficulty of gameplay.	All unit tests passed. Game difficulty was implemented as an enum to make it easy to change difficulty, and a game difficulty slider was added to the settings menu.
UR_SCREEN_SCALABILITY, NFR_SCREEN_SCALABILITY	We ran manual tests to ensure that the screen scaled appropriately with no screen tearing or off-screen components.	With the libgdx built-in FitViewport, the screen scaled successfully in the game view with no issues.
UR_PAUSABLE, FR_PAUSABLE	We wrote unit tests to ensure the game did not change update when paused, and ran manual tests to check the game could be paused at any time	All unit tests passed. We implemented the pause menu to display paused when ‘P’ was pressed
UR_PLATFORM_COMPATIBLE, NFR_PLATFORM_COMPATIBILITY	We ran manual tests across all platforms with our final JAR to confirm the game was cross OS compatible	The game ran as expected on all machines
UR_NOT_FRUSTRATING	We ran user tests and asked if the game was frustrating or fun to play	More than 70% of users said the game was fun to play and not frustrating
UR_UNIVERSITY_ACCURATE, NFR_UNIVERSITY_ACCURATE_LOOK	We ran user tests and asked if the game felt university accurate.	Many people said they liked the feel, but it could have some more university landmarks. We added a long boi status, the Nisa, the lake, and Central Hall to fix this.
FR_EASY_MOVEMENT_CONTROLS	We ran user tests and asked if the game was easy to play.	We included arrow keys in the game controls and a controls screen in the introduction. 90% of people reported they knew how to control the player.
UR_UNIVERSITY_RELATABLE_EVENTS	We ran user tests and asked if the game’s events felt university relatable	80% of people rated the game 8/10 for relatability, and we added the friend event for further relatability
NFR_CLEAR_BOUNDARIES	We ran end-to-end user tests and asked if the game had clear boundaries	80% of user testers had no issue completing the game and reported that the boundaries were easy to see
NFR_CASUAL_FRIENDLY_DIFFICULTY	We ran end-to-end user tests and asked if the game was easy to complete	80% of user testers had no issue completing the game, and 70% said it was well-balanced, possibly a bit too fast when you get a few speed boosts.

No failed tests in the current implementation

References

BBFC. “BBFC PG Rating.” *BBFC* | *PG*, https://www.bbfc.co.uk/rating/PG. Accessed 30 December 2025.